A Fast Algorithm for Hierarchical Text Classification

Wesley T. Chuang[1,3], Asok Tiyyagura[2], Jihoon Yang[3], and Giovanni Giuffrida[1]

[1] Computer Science Department, UCLA, Los Angeles, CA 90095, USA
{yelsew, giovanni}@cs.ucla.edu
[2] Department of Computer Science, Iowa State University, Ames, IA 50011, USA
asokt@cs.iastate.edu
[3] HRL Laboratories, LLC, 3011 Malibu Canyon Rd, Malibu, CA 90265, USA
{yelsew, yang}@wins.hrl.com

Abstract. Text classification is becoming more important with the proliferation of the Internet and the huge amount of data it transfers. We present an efficient algorithm for text classification using hierarchical classifiers based on a concept hierarchy. The simple TFIDF classifier is chosen to train sample data and to classify other new data. Despite its simplicity, results of experiments on Web pages and TV closed captions demonstrate high classification accuracy. Application of feature subset selection techniques improves the performance. Our algorithm is computationally efficient being bounded by $O(n\ log\ n)$ for n samples.

1 Introduction

As the amount of on-line data increases by leaps and bounds, the design of an efficient algorithm or an approach to accessing the data (e.g. through classification, clustering, filtering, etc.) has become of great interest. Two important aspects motivate such design. First, the data needs to be arranged efficiently. For example, instead of placing all the data in a flat directory, we can arrange it hierarchically based on a concept hierarchy (see Yahoo, US Patent databases, CNN and other major Internet news directories [13, 9, 2]). Querying with respect to a concept hierarchy is significantly more efficient and reliable than searching for specific keywords since the views of the data collected are refined as we go down the hierarchy [1]. Second, when text classification is our chosen approach, an efficient algorithm should be used. A number of algorithms have been proposed and their performances are compared in the literature [14].

We use the *TFIDF* text classifier [11, 4] and proceed with the following steps for hierarchical classification: The first step is to define the concept hierarchy using domain knowledge and to collect text data corresponding to the concept hierarchy. The data is then used for training the classifier and for testing the performance of our classification system. The next step is to convert the data into an appropriate form for classification (e.g. into a bag-of-words representation). Then we can derive hierarchical classifiers by supervised learning with the data collected. Finally, the classifiers are used to classify new data. Each of these steps

is described in detail in the following sections, followed by experimental results and conclusion.

2 Concept Hierarchy

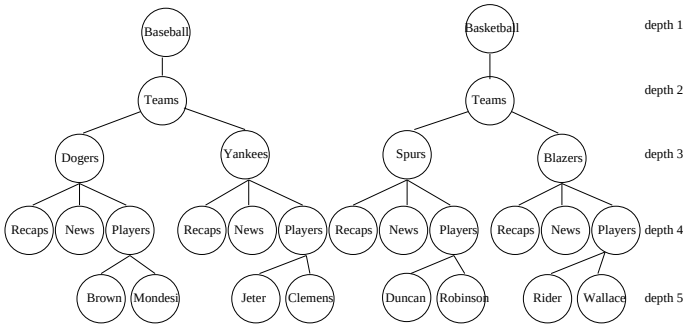


Fig. 1. A sample concept hierarchy for professional baseball and basketball.

Figure 1 shows a concept hierarchy which categorizes Web news reports for professional baseball and basketball. Initially, a concept graph is generated based on some domain knowledge. Each node in the hierarchy contains several text documents whose topic is identified as the concept.

As in relational and object-oriented databases, in which there are no absolutely standard schema (tables and classes), we do not have a faultless concept hierarchy. In fact, we should not look for one that is exclusively superior to any other even for the same data. We need one that can help us in alleviating our humans semantic burden (since a concept hierarchy encapsulates semantics) and facilitate achieving efficiency in data arrangement and searching capability.

However, unlike relational databases and object-oriented databases, Web documents, are generally unstructured. Therefore, the way we describe their schema is different from the relational and object-oriented models. Because they are unstructured, a convenient way to describe a concept in our hierarchy is to use a collection of words (features) from a document. Since it is usually very easy for humans to come up with some top-level concepts as the schema, we propose a human-generated initial concept hierarchy.

Before taking this concept hierarchy to perform feature extraction and classification, there are several assumptions to be made. First, we assume that an initial concept hierarchy has been created with each node labeled by one or a few terms representing the concept. Second, we assume that several documents have been manually placed into every node, serving as the training and testing data for *supervised learning*. Our third assumption states that a parent node owns the union of the documents of its child nodes.

3 Training the Hierarchy

After the initial hierarchy is set up and with training and testing documents placed into each node, we can proceed to "train" the hierarchy. The rationale is as follows: We will characterize those training documents residing in each node; find a range (or threshold) of the various characteristics for the node so that it can classify the test documents into certain concepts - that is, find suitable nodes to index the new documents, from top to bottom, in the concept hierarchy.

Ideally, we would hope to characterize documents in each node with a "term" or a label. However, this is not practical because for text documents, one feature term will not suffice in describing the whole document. Instead, we must "understand" the documents in some way. In the absence of a satisfactory solution to the natural language understanding problem, most current approaches to document retrieval use a *bag-of-words* representation of documents [11, 4]. It is for this reason we opt for *surface parsing* and obtain a vector (a set of word features) of weights for each document with time complexity in the order of $O(n)$, where n is the number of words in the document.

Specifically, we restrict ourselves to a relatively simple yet effective approach based on the *TFIDF* (*term frequency* $\times$ *inverse document frequency*) classifier [11, 4]. Since we are trying to separate documents into distinguishable concepts, intuitively, the combined effect of term frequency and inverse document frequency can distinguish two different document types well. Definitions of term frequency and inverse document frequency are to follow.

3.1 Representing Features for Each Node

To convert a collection of documents in each node into special representation, let's examine the documents closely. First, every document is processed using *stopping* and *stemming* procedures [11, 4] to obtain a bag of words. Stopping is the procedure to eliminate common words from the text, and stemming is the procedure to find a unique representation (e.g. root) for a word. After these procedures, we consider the following frequencies in the documents.

- *Term Frequency*: Let W be the set of words from all documents. The *term frequency* of the word w_i (ith vocabulary in W), $TF(w_i, d)$, is the number of times w_i occurs in document d.
- *Document Frequency*: $DF(w_i)$ is the number of documents in which word w_i occurs at least once.
- *Inverse Document Frequency*: $IDF(w_i) = log(\frac{|D|}{DF(w_i)})$, where $|D|$ is the total number of documents among sibling nodes.
- *Term Frequency* $\times$ *Inverse Document Frequency*: $TFIDF(w_i, d) = TF(w_i, d) \times IDF(w_i)$

We subsequently merge all child feature vectors to obtain the TF vector for a node. Calculation of TF vectors continues bottom-up until we reach the root. Meanwhile, we also propagate the DF vector for each node all the way up to the

root. When this merging process is completed, the feature vector in each node is given by: $\boldsymbol{F} =< TFIDF(w_1, d), TFIDF(w_2, d), \ldots, TFIDF(w_{|W|}, d) >$, where d is the union of documents belonging to the node, and w is the union of the set of words in d.

3.2 Determining the Threshold for Each Node

Having organized training documents into different nodes (classes) and having built feature vectors for each of them, we characterize each class by a TFIDF vector. In other words, TFIDF will now serve as a *norm* or *prototype* vector to describe that class.

Formally, let C be a collection of document classes of interest. A prototype vector $\boldsymbol{c}$ (for each class in the concept hierarchy) is generated for each class $c \in C$ as follows: $\boldsymbol{c} = \sum_{F \in c} \boldsymbol{F}$.

We will use these training documents again to make a complete hierarchical classifier in two ways. First, for each prototype vector $\boldsymbol{c}$, we need to introduce a *threshold* θ - a distance measure to indicate at what distance range to the prototype vector we consider documents fall into the same category (i.e. class). This becomes clear if we imagine that a TFIDF vector is lying in an n-dimensional hyper-space[1], shown in Figure 2. As can be seen, the threshold makes a boundary for a class. For simplicity, we visualize it after normalization (i.e. as a unit-length vector) in a 3-D coordinate system. Second, we need to examine how good the hierarchical classifier is. That is, given the prototype vectors and thresholds, what is the *accuracy* with which the classifier will correctly classify documents? Later, we will check the accuracy for other brand new test documents. However, in this section, we only consider the accuracy for the training documents which we used to come up with the prototype vectors.

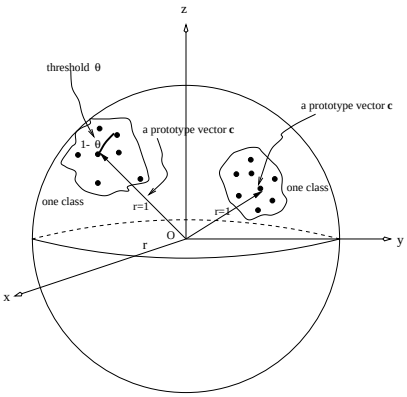


Fig. 2. Learning the threshold to admit new documents into a class.

Deriving threshold and computing training accuracy are inter-related. Here we use classification training accuracy feedback to adjust the threshold. In other words, we adjust the threshold in such a way that all documents considered for the node will yield the best accuracy. Accuracy, as shown in the following, concerns the percentage of documents correctly classifying into a class:

$$Accuracy = \frac{\#\ of\ documents\ correctly\ categorized}{\#\ of\ documents\ considered} = \frac{a+d}{a+b+c+d}$$

[1] Elements of the TFIDF vector are positive real numbers, so they can only lie in the first quadrant.

In the equation, we use a, b, c, d to indicate respectively the number of documents that should be in a class and are selected; the number selected but that should not be in; the number that should be in but are rejected; and the number that should not be in and are not selected.

Now, starting from the highest level of the hierarchy, we compare the training documents $\boldsymbol{n}$ (in vector representation) with the prototype vector of the node. When we compare two feature vectors, a *cosine* function is commonly used for similarity measure. That is, we compute the similarity of a document $\boldsymbol{n}$ to prototype vector $\boldsymbol{c}$ by: $\forall c \in C\ cos(\boldsymbol{n}, \boldsymbol{c}) = \forall c \in C\ \frac{\boldsymbol{n} \cdot \boldsymbol{c}}{\|\boldsymbol{n}\| \cdot \|\boldsymbol{c}\|}$.

```
for (level = 1 to depth of hierarchy)
   for (every node j in level)
      F_n := getPrototypeVec(Node_j);
      Docs := getDocuments(
         getSiblings(Node_j) ⋃
         getChildren(Node_j));
      < F_d > := getFeatures(Docs);
      < Dist > := cos(F_n, < F_d >);
      θ := Find a cut d in < Dist >
           that maximizes accuracy;
   end
end
```

Fig. 3. Using accuracy feedback to adjust the threshold.

For every node, we consider those training documents belonging to all sibling and child nodes. We first sort them by their distances (*Dist*) to the prototype vector. We then choose one distance that renders the best accuracy. This is the threshold, θ. Pseudocode for computing θ is shown in Figure 3. (All the functions are self-explanatory.)

3.3 Time Complexity Analysis

Our approach involves surface parsing, i.e., obtaining features for words in the documents as well as training features in the concept hierarchy. In parsing, time is linear. Most of the time is taken up by training, especially when computing the thresholds. Assume that n is the total number of documents, m is the total number of nodes in the hierarchy, and $m << n$. The contributing factors timewise are: computing TF takes $O(n)$; computing IDF takes $O(n)$; computing threshold θ takes $O(n\ log\ n)$, where $n\ log\ n$ is what the sorting costs timewise. All together, the time remains bounded by $O(n\ log\ n)$, which upholds our claim that this classification algorithm is fast.

4 Testing the Hierarchy

In this section, we test the performance of our hierarchical classifier using a test set of documents which is different from the training set. The ratio of the total number of testing documents to the training documents is about 1 to 2.

4.1 Every Node as a Classifier

When each node is equipped with a TFIDF vector and a threshold (θ), we regard the node as a *classifier*. Again, starting from the highest level of the hierarchy,

we compare the test document $\boldsymbol{n}$ (in vector representation) with the prototype vector of the node. If these two vectors are close enough, based on the cosine measure, we treat the test document as belonging to the class that the node represents, and we continue to compare the document with its child nodes. If we do not have a close match, we conclude that the document does not belong to the class and stop.

In our hierarchical classifier, a document is assigned to all the classes whose prototypes are *sufficiently* close to it. The pseudocode in Figure 4 summarizes the classification process.

Let $\boldsymbol{n}$ be the vector to be classified.
$Class = \phi$
$NodeSet = root$
while $(NodeSet \neq \phi)$ **do**
  Retrieve a $node_i$ from $NodeSet$:
    $NodeSet = NodeSet - node_i$.
  Compute the prototype vector:
    $\boldsymbol{c} = TFIDF(node_i)$.
  Compute the similarity between $\boldsymbol{n}$
    and $\boldsymbol{c}$: $s = cos(\boldsymbol{n}, \boldsymbol{c})$.
  if $(s > \theta)$
    $Class = Class \bigcup \{ c \}$.
    $NodeSet = NodeSet \bigcup$
      $\{ getChildren(node_i) \}$.
end
return $Class$

Fig. 4. Finding the class for a new text document.

4.2 Test Accuracy

Test accuracy is defined as the ratio of the number of correctly classified documents to the number of documents that are filtered at each node, i.e.,

$$test\ accuracy = \frac{\#\ correctly\ classified}{\#\ of\ filtered}$$

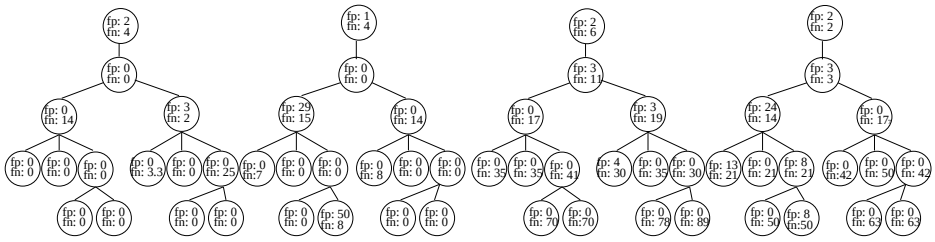
Since the classification is performed from the top to the bottom of the hierarchy, some test documents "filter" through certain nodes and continue to drill down in the hierarchy; others are stopped when their similarity measures do not pass the threshold of the prototype vectors. Consequently, the number of filtered documents diminishes as the testing process continues to the bottom level.

5 Experiments

For training and testing, we collected approximately 200 documents on professional baseball and basketball news. Experimental results demonstrate the feasibility of our approach with good classification accuracy. In training, most upper level nodes (classifiers) receive ratings of above 90% for training accuracy, while some lower level nodes perform on the average at a rate of 75%. In testing, the accuracy performance is relatively poorer, especially in lower nodes but acceptable in higher level nodes. This is because the concepts become more specific as we go down the hierarchy. In addition, the number of documents considered is relatively small compared to those at higher levels.

To account for the classification error, both in training and testing phases, we look into two different types of errors: false-positive (FP) and false-negative (FN). They are defined as follows (using a, b, c, and d as explained before):

$false\text{-}positive(FP) = \frac{\#\ mistakenly\ put\ on}{\#\ not\ about\ the\ topic} = \frac{b}{b+d}$, $false\text{-}negative(FN) = \frac{\#\ mistakenly\ missed}{\#\ about\ the\ topic} = \frac{c}{a+c}$. Figure 5 reports such a breakdown of classification errors for training and testing in the hierarchy. In the testing phase, a few lower level nodes resulted in relatively high *FN*. This is due to the depth of the nodes and the limited number of documents tested in those nodes.



(a) Training phase. (b) Testing phase.

Fig. 5. False-positive (fp) and false-negative (fn) errors (in percents).

5.1 Feature Subset Selection

The above experiment has shown that the TFIDF feature is a reliable indicator for categorizing Web text. We now investigate whether a subset of these features can perform as well in text classification.

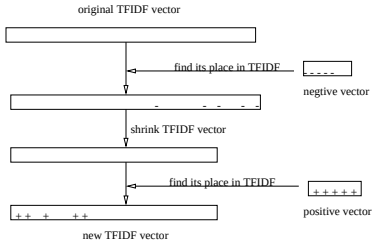
Straight TFIDF Subset One feature selection represents each node with only a subset of word features. The new feature vector for each node consists of the top m leading TFIDFs after sorting the original vector by its TFIDF values: $\boldsymbol{F}_m =< TFIDF(w_1,d), TFIDF(w_2,d), \ldots, TFIDF(w_m,d) >$, where $m <= |W|$.

Figure 6 illustrates the average accuracy (in percents) for classifiers at each depth for both training and testing phases. In testing, 10 different subsets are compared. According to our findings, when *TFIDFs* are cut down by 40-50%, they scarcely affect the test accuracy. This has two implications: First, not only is TFIDF a good indicator in text classification, but those higher TFIDFs values are the dominating terms. Second, using subset features can reduce TFIDFs storage requirements drastically and improve efficiency without compromising classification accuracy.

Special Positive/Negative Vectors Another way of selection is to introduce two special vectors (one positive, one negative) to represent the background knowledge. The positive and negative vectors are manually made. They are the human version of TFIDFs because most positive terms may coincide with the

		Average accuracy for classifiers at each depth of the hierarchy										
		Training	Testing with a subset of Tfidf features									
			10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
d	1	97%	65%	70%	77%	81%	87%	93%	93%	93%	93%	94%
e	2	100%	57%	61%	68%	72%	78%	88%	88%	88%	90%	89%
p	3	90%	61%	72%	65%	75%	76%	79%	78%	78%	78%	76%
t	4	99%	57%	67%	68%	68%	68%	68%	68%	69%	70%	64%
h	5	98%	45%	55%	44%	45%	45%	46%	46%	48%	49%	32%

Fig. 6. Average accuracy for classifiers at each depth of the hierarchy.



(a) Design.

		training	testing
d	1	98%	96%
e	2	98%	92%
p	3	92%	82%
t	4	98%	63%
h	5	98%	53%

(b) Average accuracy.

Fig. 7. Combining background knowledge in the classification.

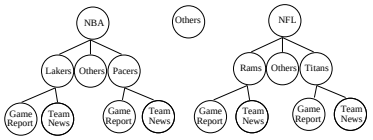
dominating TFIDFs, whereas negative terms are words that do not contribute to classification by human judgement.

Figure 7 (a) shows how this background knowledge is incorporated into the original TFIDFs for every node in the hierarchy. During training, the negative vector is subtracted from the original TFIDF vector. Later, in testing, we impose a confidence function and the positive vector on each node, and let the confidence function compete with the original similarity function. When background knowledge confidence reaches a certain level, we abandon the *cosine* measure and determine the category right away.

In this experiment, a very simple confidence function is used - the number of positive words (δ) that occur in the document. Such background knowledge somewhat mimics the the way a human determines a category. In Figure 7 (b), for example, when a δ of 4 is used, it reveals that combining such background knowledge into TFIDFs does improve the classification accuracy.

5.2 Applied to TV Closed Captions

We used the same approach as above, and applied it to TV closed caption data mixed with a few Web pages (15%). Closed captions usually contain more typos than Web pages. Even so, the classification accuracy showed some promising results (Figure 8). This is important because success in classifying TV closed captions can assist in video - or, generally multimedia - classification, in which case, time is of great concern.



(a) Concepts for TV closed caption data.

	training	testing	
		TFIDF w/ BK	BK alone
depth 1	96%	84%	81%
depth 2	91%	81%	39%
depth 3	90%	58%	20%

(b) Average accuracy.

Fig. 8. Classifying TV closed captions with TFIDF and background knowledge(BK).

6 Related Work

A number of existing approaches are similar to hierarchical classification based on a pre-defined concept hierarchy. Many of them are combined with feature subset selection which finds the best subset of features that improves classification accuracy, reduces measurement cost, storage, and computational overhead. One example, TAPER [1], makes use of a concept hierarchy and classifies text using statistical pattern recognition techniques. It finds feature subsets by the Fisher's discriminant. Similarly, Mladenic and Grobelnik proposed a document categorization method based on a concept hierarchy [8]. They used the naive Bayesian classifier on feature vector of word sequences and employed feature subsets to yield good performance. McCallum et al. also proposed a hierarchical classification using the naive Bayesian classifier [5]. In particular, they suggested combining labeled and unlabeled data to boost the classification accuracy.

There have been numerous approaches for automatic generation of a concept hierarchy. Their incentive is to eliminate the overhead of manually constructing a concept hierarchy. Sahami, for example, applied unsupervised clustering to generate a concept hierarchy from text data [10]. He made use of well-defined similarity measures to find the clusters as well as to feature subset selection. Sanderson and Croft presented a means of automatically deriving a hierarchical organization of concepts from a set of documents [12]. Their work used co-occurrence and subsumption conditions for selected salient words and phrases without standard learning or clustering techniques. In addition, there exist a variety of learning algorithms for text. Yang compared the performance of many learning algorithms [14]. Mladenic surveyed text-learning and related intelligent agents based on three key criteria: what representations to use for documents; how to select features, and what learning algorithm to use. Mitchell's book [6] is yet another comprehensive source of machine learning algorithms.

7 Summary and Discussion

The design of an intelligent text classifier is of great importance in the current world filled with such vast amounts of data. The algorithm must be fast because time is critical. To fulfill these promises, we designed and implemented a hierarchical classification system that leverages on a concept hierarchy and a simple and fast learning algorithm. The hierarchical aspect narrows down the search

space significantly by eliminating all irrelevant areas. Then, use of TFIDF and accuracy-feedback quickly makes the classifier.

Regarding experimental results, the hierarchical classifiers performed fairly well with the Web data and TV closed captions in the sports domain. Our preliminary work on feature subset selection also demonstrated improvements on classification accuracy and the cost associated with the use of features. Some avenues for future research include:

- Exploitation of data structures: Structural information in the text (e.g., title, sections, references) can be exploited and differentiated.
- Consideration of different types of data: The system can be tested with different types of text data - for instance, U.S. patent data.
- Incremental learning: New data can be learned dynamically. Meanwhile, old data should be ignored after some time has elapsed.
- Combination of labeled and unlabeled data: To reduce the overhead in data preparation prior to learning, unlabeled data can be combined with labeled data.
- Automatic expansion/shrinkage of the concept hierarchy: Dynamic change in the concept hierarchy is needed to accommodate the newly formed concepts.

References

1. S. Chakrabarti, B. Dom, R. Agrawal, P. Raghavan. Using Taxonomy, Discriminants, and Signatures for Navigating in Text Databases. In *Proceedings of the 23rd VLDB Conference*, 1997.
2. CNN.com. `http://www.cnn.com/`
3. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, S. Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the 15th Conference on Artificial Intelligence*, 1998.
4. R. Korfhage. *Information Storage and Retrieval.* New York: Wiley, 1997.
5. A.McCallum, K. Nigam, J. Rennie, and K. Seymore. Building Domain-Specific Search Engines with Machine Learning Techniques. In *AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace*, 1999.
6. T. Mitchell. *Machine Learning.* New York: McGraw Hill, 1997.
7. D. Mladenic. Text-learning and related intelligent agents: a survey. In *IEEE Intelligent Systems*, Vol.14, (no. 4), pages 44-54 July-Aug.1999.
8. D. Mladenic and M. Grobelnik. Featrure Selection for Classification based on Text Hierarchy. In *Working Notes of Learning from Text and the Web, Conference on Automated Learning and Discovery (CONALD)*, 1998.
9. US Patent and Trademark Office. `http://www.uspto.gov`
10. M. Sahami. *Using Machine Learning to Improve Information Access.* Ph.D. Dissertation, Department of Computer Science, Stanford University. 1998.
11. G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Reading, Massachusetts: Addison-Wesley, 1989.
12. M. Sanderson, B. Croft. Deriving concept hierarchies from text. In *Proceedings of the 22nd ACM SIGIR Conference*, pages 206-213, 1999.
13. Yahoo. `http://www.yahoo.com/`
14. Y. Yang. A Re-examination of Text Categorization Methods. In *Proceedings of the 22nd ACM SIGIR Conference*, pages 42-49, 1999.